第8章

インスタンスとクラス

前章では、オブジェクト指向の本質は
「現実世界の登場人物とそのふるまいを
コンピュータ内部の仮想世界で再現すること」にあると学びました。
そこで第 8 章では、実際に Java でプログラムを記述し、
仮想世界にさまざまな登場人物を生み出し、
活動させていくために必要な事柄を学んでいきます。

CONTENTS

8.1 仮想世界の作り方
8.2 クラスの定義方法
8.3 クラス定義による効果
8.4 インスタンスの利用方法
8.5 第 8 章のまとめ
8.6 練習問題
8.7 練習問題の解答

8.1 仮想世界の作り方

8.1.1 オブジェクトを生み出す手順

第７章で学んだように、オブジェクト指向プログラミングで開発されたプログラムは、動作時に「現実世界をマネた、それぞれのオブジェクトが互いに連携して動く仮想世界」を形成します。そのように考えるとプログラマの仕事は以下の２つといえます。

①各オブジェクトが負うべき責務を考え、「属性」「操作」の種類と内容を定義する。
②各オブジェクトを仮想世界に生み出し、動かす。

手順としては、①オブジェクトを定義して、②オブジェクトを生成すればいいのね。

う〜ん、惜しいね。厳密に言うと Java では「オブジェクトは定義できない」んだ。

Java では、仮想世界の中で動く**オブジェクトそのものをプログラマが直接定義することは許されません**。その代わりにプログラマは、「オブジェクトが生み出される際に用いられる、オブジェクトの設計図」である**クラス**を定義できます。

先ほどの朝香さんの言葉を訂正すると、**①クラスを定義して、②そのクラスに基づいてオブジェクトを生成する**、が正しいのです（図 8-1）。

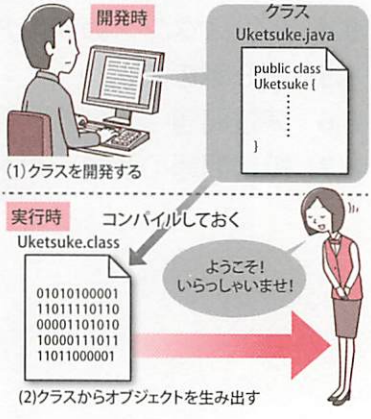


図 8-1　クラスの定義とオブジェクトの生成

なるほど。確か7章でも「開発するのはクラスで、動かすときにオブジェクトになる」って学んだわね（7.3.3項を参照）。

8.1.2 クラスとオブジェクトが別である理由

でも先輩、私やっぱり納得がいきません。オブジェクトを直接定義できたほうがシンプルでいいじゃないですか？

確かにそのほうがシンプルだね。事実、オブジェクトを直接定義・生成できる言語も存在する。だけど「クラス定義→オブジェクト生成」という方式にもメリットはあるんだ。

なぜJavaでは「オブジェクト同士が連携する仮想世界」を作るために、わざわざ「クラスを定義して、そのクラスからオブジェクトを生成する」という複雑な手順を踏まなければならないのでしょうか。その答えに辿り着くために、私たちは「**オブジェクトを大量に作る必要がある状況**」を想像しなければなりません。

たとえば銀行で「すべての口座情報の複雑な統計を計算するプログラム」が動くとき、その仮想世界には何個の「口座」オブジェクトが必要でしょうか？ 口座が1,000あるなら1,000個のオブジェクトを生み出す必要があるかもしれません。その**1,000個の口座オブジェクトそれぞれに対して、「属性として、残高・名義人・開設日があって…」という定義を繰り返す必要がある**としたら、それを作るプログラマの作業は膨大なものになります。

そこでクラスの登場です。「属性として、残高・名義人・開設日があって…」という定義をした「口座クラス」を1度作っておけば、この**クラスから100個でも1,000個でも必要な数だけオブジェクトを生み出すことができる**のです。たとえば振り込みを受け付ける「Uketsukeクラス」を1つ準備しておけば、複数の受付係を生み出して、「並行して振り込み依頼を受け付ける」プログラムを作ることもできます（次ページの図8-2）。

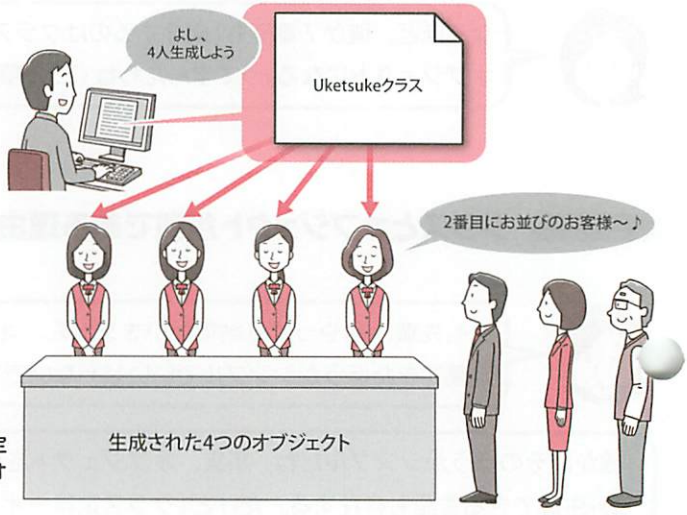


図 8-2　1 つのクラスを定義しておけば、何個でもオブジェクトを生成できる

クラスって、プラモデル工場にある「金型（かながた）」と同じと考えればいいですか？ 元となる金型を１個作っておいて、そこにプラスチックを流し込んで、同じプラモデルを大量に製造するんです。

うん。わかりやすいたとえだね。

ここで改めて意識してほしいのは、**クラスとオブジェクトはまったく違うものであるという点です。プログラムの動作時に仮想世界の中で活躍するのは「オブジェクトだけ」であって、その金型であるクラスが仮想世界で活動することは基本的にありません。**「受付係の金型」が挨拶をしたり振り込みを受け付けることはないのです。

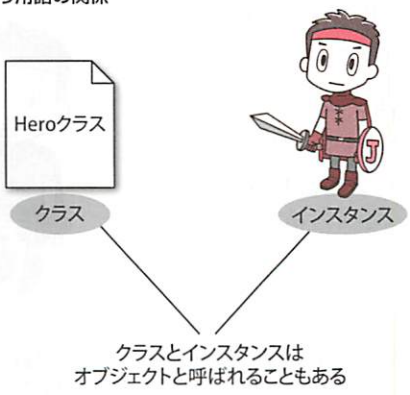
8.1.3　オブジェクトという用語のあいまいさ

実は、実際の開発現場における技術的な会話や文章の中で単に「オブジェクト」という表現が用いられる場合、「金型か、その金型から生まれた実体かを厳密に区別しない（会話の都合上、どちらでもいい）」ことがあります。つまり、「オブジェ

クト」という用語は、ときどきクラスのことを指して使われることもある、かなりあいまいなものなのです。

もし、「金型ではなく、その型から生み出された仮想世界で活動する実体」を厳密に示したい場合は、**インスタンス**（instance）という用語を用います。また、クラスからインスタンスを生成する行為を**インスタンス化**（instantiation）と表現することもあります。本書でも以降、オブジェクトのことは極力インスタンスと表現します（図 8-3）。

図 8-3　クラス・インスタンス・オブジェクトという用語の関係



インスタンスとクラスの関係

仮想世界で活動するのは「**インスタンス**」であり、そのインスタンスを生み出すための金型が「**クラス**」である。

8.1.4 プログラムに登場する 2 種類のクラス

それではいよいよ、第 7 章で登場した「勇者とお化けキノコの戦い」プログラムを作っていこう。

やったー！ 勇者クラスと、お化けキノコクラスを作るんですよね？

いや、実はもう 1 つクラスを作る必要があるんだ。

第７章で紹介した「勇者とお化けキノコの戦い」をプログラムにするにあたって、まず必要なものとして、勇者（Hero）クラスとお化けキノコ（Matango）クラスはすぐに思いつくでしょう（これらのクラスは「現実世界の登場人物に対する金型」ですので、以後「登場人物のクラス」と表現します）。

図 8-4　勇者やお化けキノコに指示を送る人が誰もいない

しかし、このような「登場人物のクラス」だけではプログラムは動きません。クラスから生み出された勇者やお化けキノコのインスタンスは、**誰かから指示（操作の呼び出し）をもらうことで責任を果たすために動く**ので、この２人だけを開発しインスタンス化しても、いわゆる「指示待ち状態」になってしまうのです（図 8-4）。

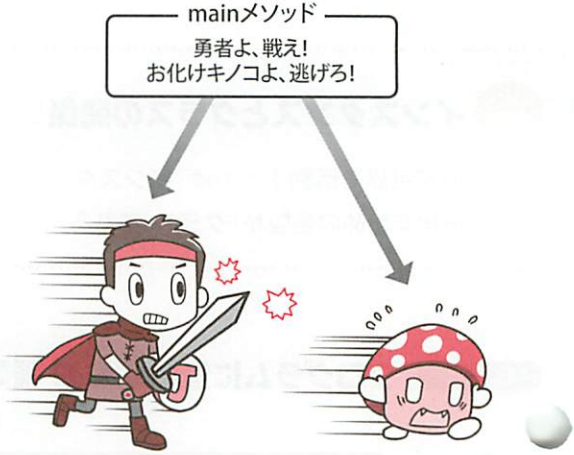


図 8-5　勇者やお化けキノコに指示を送る main メソッド

そこで勇者とお化けキノコを作った私たち「神様」が、この２人の登場人物にどのように動くかを指示していく必要があります。私たち神様の指示である「天の声」は、main メソッドとして記述します（図 8-5）。

ただ単に main メソッドを記述したいだけだとしても、メソッドはクラスの中に作るという Java のルールを守る必要があります。よって、Main などの適当な名前でクラスを１つ作り、その中に main メソッドを作ることになります。

リスト 8-1　main メソッドを作る

Main.java

```
1  public class Main {
2    public static void main(String[] args) {
3      //  (以下の内容をJavaで記述していく)
4      // 勇者よ、この仮想世界に生まれよ！
5      // お化けキノコよ、この仮想世界に生まれよ！
6      // 勇者よ、戦え！
7      // お化けキノコよ、逃げろ！
8    }
9  }
```

そして、以下のようにして Main クラスの main メソッドを起動します。

```
>java Main↵
```

これで、勇者やお化けキノコが仮想世界で動き出します。

原理的には、勇者クラスやお化けキノコクラスのどちらかのクラスの中に main メソッドを同居させることもできる。しかし、コードがわかりにくくなってしまうため、Main クラスのように独立したクラスを準備して、その中に作ることをお薦めするよ。

この Main クラスだけは「現実世界の登場人物を模したものではない」ですし、インスタンス化して利用するものでもありません。あくまでも仮想世界の神様として、それぞれの登場人物を生み出し、それらに対して指示を出すことが役割です（本書では、このようなクラスを「神様のクラス」と表現します）。プログラムを作る場合には、「登場人物のクラス」と「神様のクラス」の 2 種類を作る必要があることを意識しておきましょう。

Java プログラムの組成に必要なクラスたち

・main メソッドを含む、1 つの「神様のクラス」
・現実世界の登場人物を模した、複数の「登場人物のクラス」

たとえば「勇者とお化けキノコの戦い」では、次の図 8-6 のように最低でも 3 つのクラスの開発が必要になるでしょう。

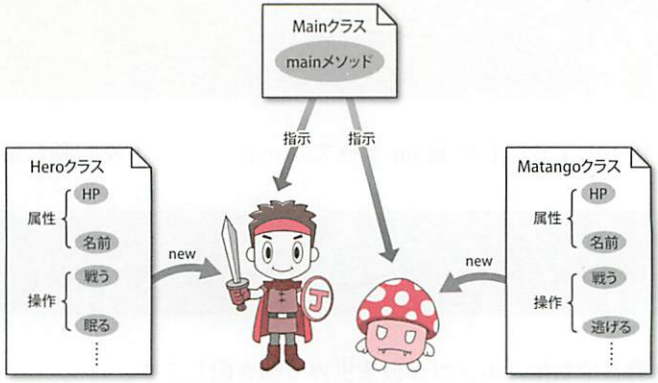


図 8-6　プログラム「勇者とお化けキノコの戦い」全体の構図

以降の解説では、その進行に従って扱うクラスが変わっていきます。その際には「登場人物クラス」の解説なのか、「神様クラス」の解説なのかを意識しながら読み進めるのがスムーズな理解のポイントです。

8.2 クラスの定義方法

8.2.1 登場人物クラスの作り方

それでは、Hero や Matango クラスを作るために、「クラスの定義方法」を学んでいきましょう。前節で学んだように、クラスには「どのような属性や操作を持っているか」を記述していきます。現時点で私たちが思い描いている「勇者クラスが持つべき属性や操作」は、右の図 8-7 のようなものです。

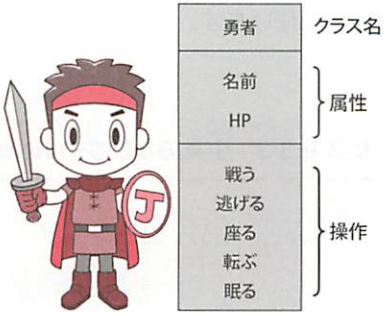


図 8-7　勇者クラスの基本設計

なお、この図のように、あるクラスの設計内容を上から「クラス名」「属性」「操作」の一覧として並べる書き方は、クラス図（class diagram）という設計図のルールに準じたものです。クラス図は世界共通の設計図として定義された UML（Unified Modeling Language）で定められている図の 1 つです。この基本設計に基づき、実際に Java でプログラムを書いていくと以下のようなコードになります。

リスト 8-2　Hero クラスを Java のコードで表したもの

Hero.java

```
1  public class Hero {
2    String name;
3    int hp;
4    void attack() {…}
5    void run() {…}
6    void sit(int sec) {…}
7    void slip() {…}
8    void sleep() {…}
9  }
```

属性の定義

操作の定義

ここからは、「クラス名」「属性」「操作」が、どのようにしてコードに書き換わったか、順を追って説明しましょう。

8.2.2 クラスの宣言方法

クラスの宣言文法自体は、第Ⅰ部で学習したものと変わりません。class キーワードを使います。次のリスト 8-3 は中身のない空の Hero クラスを作ってみたものです。

リスト 8-3　中身のない空の Hero クラスを作成

Hero.java

```
1  public class Hero {
2  }
```

第 1 章で学習した「Java のファイル名の約束事」（1.2.1 項）に従い、この場合のファイル名は Hero.java にする必要があることを忘れないようにね。

8.2.3 属性の宣言方法

図 8-7 に示したクラス図によれば、勇者は「名前」と「HP」の 2 つの属性を持っています。そこで、これらの属性について、プログラムで使用する変数の名前と型を考えます。名前は文字列ですから String 型ですね。同様に HP は数値ですので int 型になります。

【名前】　name（String 型）
【HP】　hp（int 型）

これらの変数を、リスト 8-3 の Hero クラスのブロック内に宣言したものが次のコードです。

リスト 8-4　Hero クラスに名前と HP を変数として宣言

Hero.java

```
1  public class Hero {
2    String name;  // 名前の宣言
3    int hp;       // HPの宣言
4  }
```

宣言を追加

上記のようにクラスブロック内に宣言された変数を、Java では特にフィールド(field)といいます。これで、name と hp という 2 つのフィールドの宣言が完了したわけです。

フィールドの宣言

属性を宣言するにはクラスブロックの中に変数宣言を記述する。

8.2.4 属性の初期値指定と定数フィールド

ところで、次のようにフィールド宣言と同時に値の代入も行うよう記述しておくと、そのフィールドの初期値を設定することができます(1.3.4 項を参照)。

リスト 8-5　フィールドを宣言すると同時に初期値も設定

Matango.java

```
1  public class Matango {
2    int hp;
3    int level = 10;
4  }
```

お化けキノコのレベルに初期値 10 を設定した

さらに、フィールド宣言の先頭に final を付けると、値を書き換えることのできない定数フィールドになります(1.3.5 項を参照)。なお、定数フィールドの名前は一目でそれとわかるように大文字で記述することが推奨されます。

リスト 8-6　フィールドを定数として宣言

Matango.java

```
1  public class Matango {
2    int hp;
3    final int LEVEL = 10;
4  }
```

フィールド LEVEL は 10 で固定

8.2.5 操作の宣言方法

では次に、Hero クラスに「操作」をプログラミングしていきましょう。とはいえ、クラス図のすべての操作を一度に作ることは大変なので、まずは「眠る」操作だけを作ります。「操作」をプログラミングするには、まず「操作の名前」「操作するときに必要な情報の一覧」「操作の結果として指示元に返す情報」「処理内容」の 4 つを考える必要があります。具体的な要素としては以下のようになります。

【名前】　sleep
【必要情報】　なし
【結果】　なし
【処理内容】「眠った後は HP が 100 に回復する」

これらを Hero クラスのクラスブロック内に記述すると次のようになります。

リスト 8-7　「眠る」操作に含まれる要素を記述

Hero.java

```
1  public class Hero {
2    String name;
3    int hp;
4
5    void sleep() {
6      this.hp = 100;
7      System.out.println(this.name + "は、眠って回復した！");
8    }
9  }
```

自分自身の hp フィールド

自分自身の name フィールド

この部分を追加

これ、第Ⅰ部でも使っていた「メソッド」じゃないですか？

そうだよ。オブジェクト指向では、ある登場人物の操作を定義するためにメソッドを使うんだ。

メソッド内部の「this.hp」や「this.name」は初めて見る記述ですね。this とは特別に準備された変数で、「自分自身のインスタンス」を意味しています。また、ドット (.) には、日本語でいう「の」と同じ意味がありますので (6.1.2 項)、「this.hp = 100;」は、「自分自身のインスタンスの hp フィールドに 100 を代入」という意味になります。これで、Hero クラスに sleep() メソッドを宣言することができました。

this は省略しないで!

同じクラス内のフィールドへのアクセスの場合、「this.」を省略しても動作します。たとえば「this.hp = 100;」と「hp = 100;」は同じ動作です。しかし、ローカル変数や引数にも同じ hp という名前の変数がある場合、そちらが優先されてしまうなど予想外の動作になる可能性があります。フィールドを用いるときには明示的に this を付けるようにしましょう。

8.2.6 クラス名とメンバ名のルール

クラス名、フィールド名、メソッド名はそれぞれ、基本的に識別子のルール (1.3.2 項) を満たしていれば自由に決めることができます。しかし、実際には次のような慣例に従って名前を付けることが望ましいとされます。

クラス名	名詞	単語の頭が大文字	Hero、MonsterInfo
フィールド名	名詞	最初以外の単語の頭が大文字	level、itemList
メソッド名	動詞	最初以外の単語の頭が大文字	attack、findWeakPoint

8.2.7 クラス定義のまとめ

ここまで、Hero クラスを題材にクラスの定義方法を学びました。クラスを定義するには、まず「属性」「操作」の一覧をクラス図にまとめ、それを Java のコードに置き換えて「フィールド」と「メソッド」として記述していきます（図 8-8）。

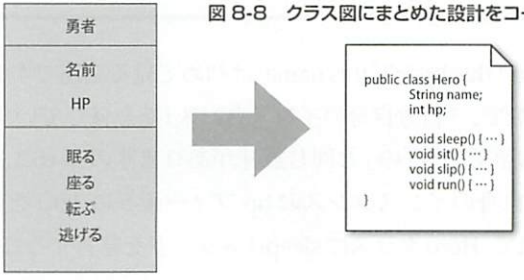


図 8-8　クラス図にまとめた設計をコードに落とし込む

この考え方に沿って、勇者クラスに「座る」「逃げる」「転ぶ」メソッドを追加したものが次のコードです。

リスト 8-8　メソッド「座る」「逃げる」「転ぶ」を追加

Hero.java

```
1  public class Hero {
2    String name;
3    int hp;
4    void sleep() {
5      this.hp = 100;
6      System.out.println(this.name + "は、眠って回復した！");
7    }
8    void sit(int sec) {
9      this.hp += sec;
10     System.out.println
           (this.name + "は、" + sec + "秒座った！");
11     System.out.println("HPが" + sec + "ポイント回復した.");
12   }
13   void slip() {
```

sleep メソッド

何秒座るか引数で受け取る

座る秒数だけ HP を増やす

slip メソッド

```
14      this.hp -= 5;
15      System.out.println(this.name + "は、転んだ！");
16      System.out.println("5のダメージ！");
17    }
18    void run() {          逃げる（run メソッド）
19      System.out.println(this.name + "は、逃げ出した！");
20      System.out.println("GAMEOVER");
21      System.out.println("最終HPは" + this.hp + "でした");
22    }
23  }
```

8.3 クラス定義による効果

8.3.1 クラス定義によって可能になる２つのこと

前節で私たちは、Hero クラスを定義することができました。このように、あるクラスを定義することによって、Java 言語では２つのことが可能になります。

１つ目は「**そのクラスに基づいて、インスタンスを生成できるようになる**」ことです。私たちは、そもそもインスタンスを生成するために、わざわざクラスを作ってきたわけですから、当然のことですね。

２つ目は「**そのクラスから生まれたインスタンスを入れる変数の型が利用できるようになる**」ことです。たとえば、Hero クラスを定義すると **Hero 型の変数が利用できるようになります**。このように、クラスを定義することで利用可能になる型のことを**クラス型**（class type）といいます。

Hero 型？ 変数って、数字とか文字列とかを入れるためのものですよね？

そもそもインスタンスを変数に入れることなんてあるんですか？

初めて学ぶことだから、多少の混乱も無理はないね。ではちょっと腰を据えて解説しよう。

8.3.2 クラス型変数とは

まずは湊くんの疑問に答えよう。そもそも Hero 型の変数ってどんな変数なのか、だね。

Java で扱うすべての変数は必ず何らかの**型**(type)を持っています。今まで利用してきた「整数を入れるための int 型」や、「文字列を入れるための String 型」は、**Java が標準で準備しており、いつでも使える型**でした。それに加え、たとえば Hero クラスを定義することで「Hero クラスから仮想世界に生み出されたインスタンスを入れることができる Hero 型」が使えるようになります。つまり、**クラスを定義すれば Java で利用可能な型の種類はどんどん増えていく**のです(図 8-9)。

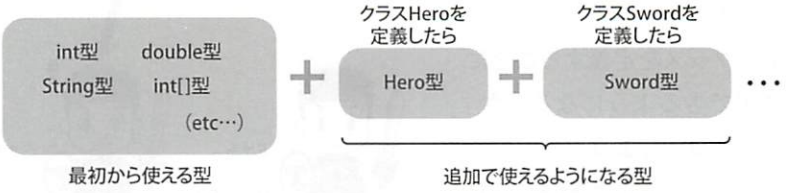


図 8-9　Java で利用できるクラスたち。新たなクラスを定義することで、その型も利用できるようになる

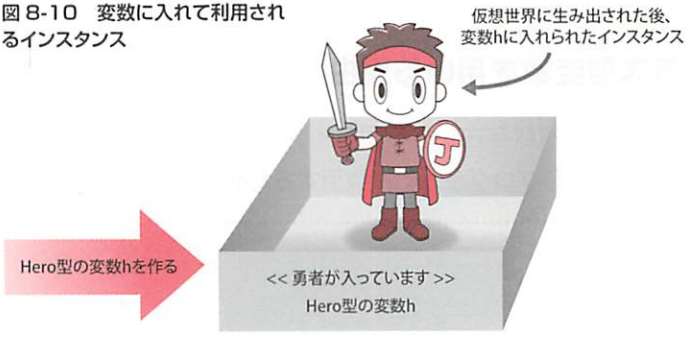
クラス型変数を準備する方法は、int 型や String 型と同じです。

リスト 8-9

```
1  Hero h;
```

この Hero 型の変数 h には、まだ勇者インスタンスは入っていません。実際には h に「仮想世界に生み出した勇者のインスタンス」を代入して利用していきます。**インスタンスは通常、クラス型変数に入れて利用する**のです(図 8-10)。

図 8-10　変数に入れて利用されるインスタンス



8.3.3 クラス型変数が必要な理由

そもそも、なぜHero型のような型にインスタンスを代入する必要があるのでしょうか？ 次のようなケースを想像すれば、「なぜJavaにはインスタンスを入れるための変数の型（＝クラス型）が必要で、クラスを定義することにより利用できるようになるのか」を理解できるでしょう。

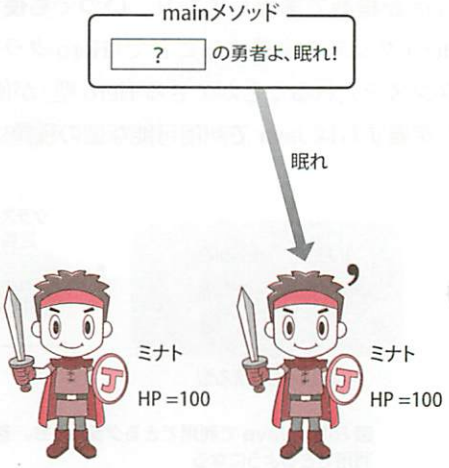


図8-11　2つの同名インスタンスを、どのように識別するのか?

図8-11を見てください。これは定義済みのクラスHeroから2つの勇者インスタンスを生み出したものですが、どちらも同じ「ミナト」という名前で、HPも同じく100です。では、この図の右側にいる勇者ミナトに「眠れ」という指示を送るには、どのようにプログラムを記述すればよいでしょうか？

紙面では「右のミナト」「左のミナト」と表現することができますが、**仮想世界の中では2人の勇者を識別する方法がない**ため、指示を送ろうにも相手を特定することができません。しかし、もし2人の勇者が、それぞれ変数h1とh2に入っているとしたら問題は解決します。「h2の勇者に眠れという指示を送る」とプログラムを書けばよいのです。

クラス型変数を用いる理由

仮想世界内に複数存在しうる同名インスタンスの中から、特定の1つのインスタンスをプログラム的に識別するため。

8.4 インスタンスの利用方法

8.4.1 「神様のクラス」の作り方

Hero クラスがやっと完成しました。でも Main クラスを作らないと。

Hero クラスだけではプログラムは動きません。勇者に指示を出す「天の声」が必要だからです（8.1.4 項を参照）。そこで、この節からは、「神様のクラス」である Main クラス（＝ main メソッド）を作っていきます。Main クラスの大まかな形は次のようなものになります。

リスト 8-10 「神様のクラス」を作る

Main.java

```
1  public class Main {
2    public static void main(String[] args) {
3      ここにプログラムの中身を書いていく
4    }
5  }
```

それでは、main メソッドの中に、以下のような内容をプログラミングしていきましょう。

1. 仮想世界に勇者を生み出す
2. 生み出された勇者に、最初の HP と名前を設定する
3. 勇者に「5 秒座れ」「転べ」「25 秒座れ」「逃げろ」と指示を出す

8.4.2 インスタンスの生成方法

では、仮想世界に勇者を生み出すために、「インスタンスの生成方法」について学びましょう。通常、Hero クラスという金型から実体のあるインスタンスを生成するには次のように記述します。

インスタンスの生成

```
クラス名 変数名 = new クラス名 ();
```

この文法に従って作成した Main クラスは次のようになります。

リスト 8-11

Main.java

```
1  public class Main {
2    public static void main(String[] args) {
3      // 1.勇者を生成
4      Hero h = new Hero();
5    }
6  }
```

Hero クラスからインスタンスを生成し、変数 h に入れる

ここで実際にインスタンスを生成しているのは、右辺の「new Hero()」という部分であり、「=」によって生成したインスタンスを Hero 型変数 h に代入しています（この過程の詳細は、次章の 9.1 節で説明します）。

なお、上記の例は 1 行で書いていますが、「Hero h;」「h = new Hero();」の 2 行に分けて記述しても、ほぼ同じ意味です。

これで図 8-10（p.315）の状態になったわけね。

8.4.3 インスタンスのフィールド利用

生み出されたばかりの勇者 h には、HP も名前もまだありませんので、それぞれのフィールドに値を代入してあげましょう。勇者 h のフィールドに値を代入するには次のような文法を使います。

フィールドへの値の代入

変数名 . フィールド名 = 値 ;

フィールド値を取得するには、単に「変数名 . フィールド名」と記述します。ここまでの内容を用いて main メソッドを改良したものが次のコードです。

リスト 8-12

Main.java

```
1  public class Main {
2    public static void main(String[] args) {
3      // 1.勇者を生成
4      Hero h = new Hero();
5      // 2.フィールドに初期値をセット
6      h.name = "ミナト";
7      h.hp = 100;
8      System.out.println("勇者" + h.name + "を生み出しました！");
9    }
10 }
```

追加した部分
変数 h の name に代入
変数 h の hp に代入
変数 h の name を取り出す

8.4.4 インスタンスのメソッド呼び出し

それでは、いよいよ勇者の冒険の始まりです。さまざまな指示を勇者に送りましょう。この「指示」は、実際には「メソッド呼び出し」という形で実現します。

リスト 8-13

Main.java

```
1  public class Main {
2    public static void main(String[] args) {
3      // 1.勇者を生成
4      Hero h = new Hero();
5      // 2.フィールドに初期値をセット
6      h.name = "ミナト";
7      h.hp = 100;
8      System.out.println("勇者" + h.name + "を生み出しました！");
9      // 3.勇者のメソッドを呼び出してゆく
10     h.sit(5);      // 5秒座れ
11     h.slip();      // 転べ
12     h.sit(25);     // 25秒座れ
13     h.run();       // 逃げろ
14   }
15 }
```

では、このコードを実行してみましょう。

```
>javac Main.java  Hero.java↵
>java Main↵
```

実行結果

```
勇者ミナトを生み出しました！
ミナトは、5秒座った！
HPが5ポイント回復した.
ミナトは、転んだ！
5のダメージ！
ミナトは、25秒座った！
HPが25ポイント回復した.
```

```
ミナトは、逃げ出した！
GAMEOVER
最終HPは125でした
```

おお！ ゲームらしくなってきましたね！

お化けキノコも生み出せば図 8-5 (p.304) のようなこともできるよ。

ここでぜひ着目してもらいたいのは、この **main メソッドの内容が「まるで冒険物語のシナリオ台本みたい」でわかりやすい**点です。もしオブジェクト指向を学習する前の私たちであれば、この画面出力結果を得るために、main メソッドの中にいくつもの HP 計算処理や System.out.println() 呼び出しを繰り返し並べていたことでしょう。たとえば次のようなプログラムです。

リスト 8-14　オブジェクト指向でない方法で作ったプログラム

Main.java

```
1  public class Main {
2    public static void main(String[] args){
3      int yusha_hp = 100;
4      int matango1_hp = 50;
5      int matango2_hp = 48;
6      String yusha_name = "ミナト";
7      int matango1_level = 10;
8      int matango2_level = 10;
9      System.out.println(yusha_name + "は5秒座った!");
10     yusha_hp += 5;
11     System.out.println("HPが5ポイント回復した!");
12     :
```

```
13          ⋮
14      }
15  }
```

リスト 8-13 をもう一度眺めてください。次のような「細かい処理」はいっさい出てきません。

HPを増やしたり減らしたりという細かい計算処理
戦闘中画面にどのようなメッセージを出すかという細かい内容

main メソッドに記述されていることは、ただ単に「勇者を登場させ、座らせ、転ばせ、座らせ、逃げさせる」ということだけです。にもかかわらず、最終的に動いた結果としては、何か行動するたびに HP の増減が処理され、画面には 10 行もの表示がなされ、最終的に HP が 125 であることが正しく出力できています。

小規模な例ではありますが、「オブジェクト指向を使うとプログラムが把握しやすくなる」ことを体験できたでしょうか。

7.4.1 項で出てきたサッカーの例で、監督がそれぞれの選手の一挙手一投足を指示しなくてよかったのと同じね。

うーん、ボクはオブジェクト指向を使わないほうがわかりやすく感じるかもなぁ…。Hero クラスを作ったり、Main クラスを作ったり、あっちこっちのソースコードを見ないといけなくて…。

それは湊くんが「複数のクラスを作っていくこと」に慣れていないからだよ。大丈夫、すぐに慣れるよ。

8.4.5 インスタンス利用のまとめ

ここまでの学習で、定義済みの Hero クラスから勇者インスタンスを生成し、そのフィールドに値を代入したり、さまざまな指示を出したりできるようになり

ました。整理すると以下のようになります。

- インスタンスの生成には new を使う。
- フィールドを利用する場合は「変数名 . フィールド名」と記述する。
- メソッドを呼び出す場合は「変数名 . メソッド名 ()」と記述する。

ここまで学んできたことを使って、仮想世界に勇者とお化けキノコ 2 匹を生み出すプログラムを書くと次のようなものになります。

リスト 8-15　お化けキノコクラスの定義

Matango.java

```
1  public class Matango {
2    int hp;
3    final int LEVEL = 10;
4    char suffix;
5    void run() {
6      System.out.println
           ("お化けキノコ" + this.suffix + "は逃げ出した！");
7    }
8  }
```

リスト 8-16 仮想世界に勇者とお化けキノコ 2 匹を生み出すプログラム

Main.java

```
1  public class Main {
2    public static void main(String[] args) {
3      Hero h = new Hero();
4      h.name = "ミナト";
5      h.hp = 100;
6
7      Matango m1 = new Matango();
8      m1.hp = 50;
9      m1.suffix = 'A';
10
```

勇者を生成し初期化

お化けキノコ A (1 匹目) を生成し初期化

```
11      Matango m2 = new Matango();
12      m2.hp = 48;
13      m2.suffix = 'B';
14
15      // 冒険のはじまり
16      h.slip();
17      m1.run();
18      m2.run();
19      h.run();
20    }
21  }
```

お化けキノコ B（2 匹目）を生成し初期化

勇者は転ぶ

お化けキノコ A が逃げる

お化けキノコ B も逃げる

勇者も逃げる

8.4.6 オブジェクト指向のクラスは現実世界とつながっている

クラスの作り方・使い方は理解できました。でも、なんだか不思議な感覚だなぁ…。クラスやメソッドって第 I 部でも使ってきましたよね？

私も。今までも多くのクラスやメソッドを作ってきたのに、この章で作ったクラスは印象がまったく違っていて、とても同じものとは思えないわ…。

この第 8 章で Hero クラスのコードを学んだ後に、第 I 部（第 1 ～ 6 章）のコードと見比べてください。**それぞれの印象が大きく違う**と思いませんか？ 第 I 部のクラス（HelloWorld など）と、第 II 部のクラス（Hero など）は、どちらも同じ「public class クラス名」で宣言されたクラスで「**文法的には両者とも正しいクラス**」です。しかし、この両者には「クラスを何のために用いるか」という**思想が伴っているか否かに決定的な違い**があります。

次ページの図 8-12 の右側にあるように、第 I 部で学んだ HelloWorld などのクラスには、何を基準にクラスとするか、何を基準にメソッドとするかという明確な思想はありませんでした。強いて挙げるなら「そろそろメソッドが大きくなっ

てきたので分割しよう」、あるいは「この機能のメソッドはこのクラスにまとめておこう」などといった、**開発者の都合や機能の単位などでクラスやメソッドを作ってきた**と言えます。

一方、図 8-12 の左側にある第 8 章の Hero クラスなどは、**「オブジェクト指向」という明確な思想に基づいてクラスやメソッドが作られています**。すなわち現実世界の登場人物を 1 つのクラスとして捉え、その登場人物の持つ属性をフィールドに、そして操作をメソッドとしてコードを書いたのです。

第 I 部では「現実とは無関係のクラス」のみを取り扱ってきましたが、読者のみなさんは、この第 8 章で初めて**「現実世界と意味がつながったクラス」**と出会いました。それが「印象が違う」と感じた理由です。

オブジェクト指向を意識したプログラム開発とは、**「現実世界の人や物、出来事をクラスに置き換えていく」**作業にほかなりません。たとえば、自動車組立て工場のシステムを作るなら「Car」や「Engine」といった部品のクラスを作ります。あるいは学生と教員を管理するプログラムなら、「Student」や「Teacher」といったクラスを作ることになるでしょう。

現実に似せて作り、現実に似せて動かしていく、これがオブジェクト指向の根本的な思想なのです。

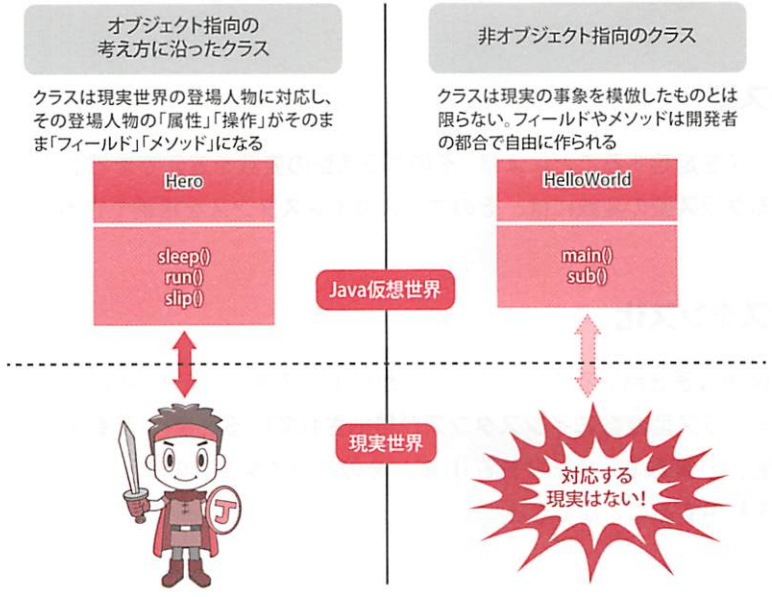


図 8-12　オブジェクト指向の考え方に沿ったクラスは現実の事柄をコードに置き換えたものである

8.5 第8章のまとめ

この章では、次のようなことを学びました。

インスタンスとクラス

- インスタンスとクラスはまったく別のものであり、混同してはならない。
- 仮想世界で活動するのは「インスタンス」(オブジェクトとも言う)。
- インスタンスを生み出すための金型が「クラス」。

フィールドとメソッド

- クラスには、属性としてフィールドを、操作としてメソッドを宣言する。
- final が付いたフィールドは、定数フィールドであり値が不変になる。
- this は「自分のインスタンス」を表すキーワードである。

クラス型

- クラスを定義することにより、そのクラス型の変数を宣言できるようになる。
- あるクラス型の変数には、そのクラスのインスタンスを格納できる。

インスタンス化

- new 演算子を用いることで、クラスからインスタンスを生み出せる。
- あるクラス型変数にインスタンスが格納されているとき、「変数名 . フィールド名」や「変数名 . メソッド名 ()」で、そのインスタンスのフィールドやメソッドを利用することができる。

8.6 練習問題

練習 8-1

現実世界の聖職者「クレリック」を表現するクラス Cleric を public で宣言してください。その際、属性や操作は宣言する必要はありません（中身が何もないクラスで構いません）。

練習 8-2

聖職者は勇者のように名前や HP を持っており、さらに魔法を使うための MP も持っています。そこで、練習 8-1 で宣言した中身がない Cleric クラスに「名前」「HP」「最大 HP」「MP」「最大 MP」を属性として追加してください。なお、HP と最大 HP は整数で初期値 50、MP と最大 MP は整数で初期値 10 であり、最大 HP と最大 MP は定数フィールドとして宣言してください。そして作成したクラスに適切なファイル名を付けてください。

練習 8-3

聖職者は「セルフエイド」という魔法を使うことができ、MP を 5 消費することで自分自身の HP を最大 HP まで回復することができます。そこで、練習 8-2 で宣言した Cleric クラスに、「selfAid()」というメソッドを追加してください。なお、このメソッドは引数なしで、戻り値もありません。

練習 8-4

聖職者は「祈る」（pray）という行動を取ることができ、自分の MP を回復できます。回復量は祈った秒数にランダムで 0 ～ 2 ポイントの補正を加えた量です（3 秒祈ったら回復量は 3 ～ 5 ポイントのいずれか）。ただし最大 MP よりも回復することはありません。

そこで、練習 8-3 で宣言した Cleric クラスに「pray()」というメソッドを追加してください。このメソッドは引数に「祈る秒数」を指定することで、戻り値として「実際に回復した MP の量」を返します。

8.7 練習問題の解答

練習 8-1 の解答

クラスの宣言に関する問題です。正解のコードは以下のとおりです。

Cleric.java

```
1  public class Cleric {
2  }
```

練習 8-2 の解答

フィールドの宣言に関する問題です。以下は解答例のコードです（おおむね合っていれば正解とします）。

Cleric.java

```
1  public class Cleric {
2    String name;
3    int hp = 50;
4    final int MAX_HP = 50;
5    int mp = 10;
6    final int MAX_MP = 10;
7  }
```

このコードのファイル名は「Cleric.java」になります。

練習 8-3 の解答

メソッドの宣言に関する問題です。以下は解答例のコードです（おおむね合っていれば正解とします）。

Cleric.java

```
1  public class Cleric {
2    String name;
```

```
  int hp = 50;
  final int MAX_HP = 50;
  int mp = 10;
  final int MAX_MP = 10;

  public void selfAid() {
    System.out.println(this.name + "はセルフエイドを唱えた！");
    this.hp = this.MAX_HP;
    this.mp -= 5;
    System.out.println("HPが最大まで回復した。");
  }
}
```


練習 8-4 の解答

引数と戻り値があるメソッドの宣言に関する問題です。以下は解答例のコードです（おおむね合っていれば正解とします）。

Cleric.java

```
import java.util.*;
public class Cleric {
  String name;
  int hp = 50;
  final int MAX_HP = 50;
  int mp = 10;
  final int MAX_MP = 10;

  public void selfAid() {
    System.out.println(this.name + "はセルフエイドを唱えた！");
    this.hp = this.MAX_HP;
    this.mp -= 5;
    System.out.println("HPが最大まで回復した。");
  }
```

```
15
16    public int pray(int sec) {
17      System.out.println
            (this.name + "は" + sec + "秒間天に祈った！");
18
19      // 論理上の回復量を乱数を用いて決定する
20      int recover = new Random().nextInt(3) + sec;
21
22      // 実際の回復量を計算する
23      int recoverActual = Math.min(this.MAX_MP - this.mp, recover);
24
25      this.mp += recoverActual;
26      System.out.println("MPが" + recoverActual + "回復した。");
27      return recoverActual;
28    }
29  }
```